

What is Git?

Git is a version control tool for software development. To put it simply, code changes a lot, and Git helps developers track, monitor, and manage those changes.

What is GitHub?

GitHub is a platform where developers can collaborate and do version control on their software. When you get the hang of it, it'll be easier to work together on group projects. You won't need to use Google Drive or send code using Facebook Messenger.

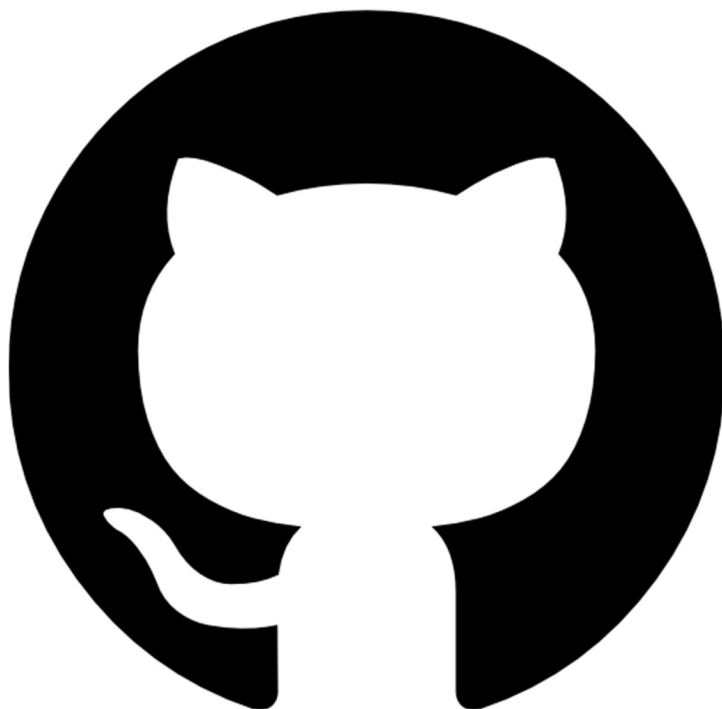
For context, here are some things GitHub has helped me with:

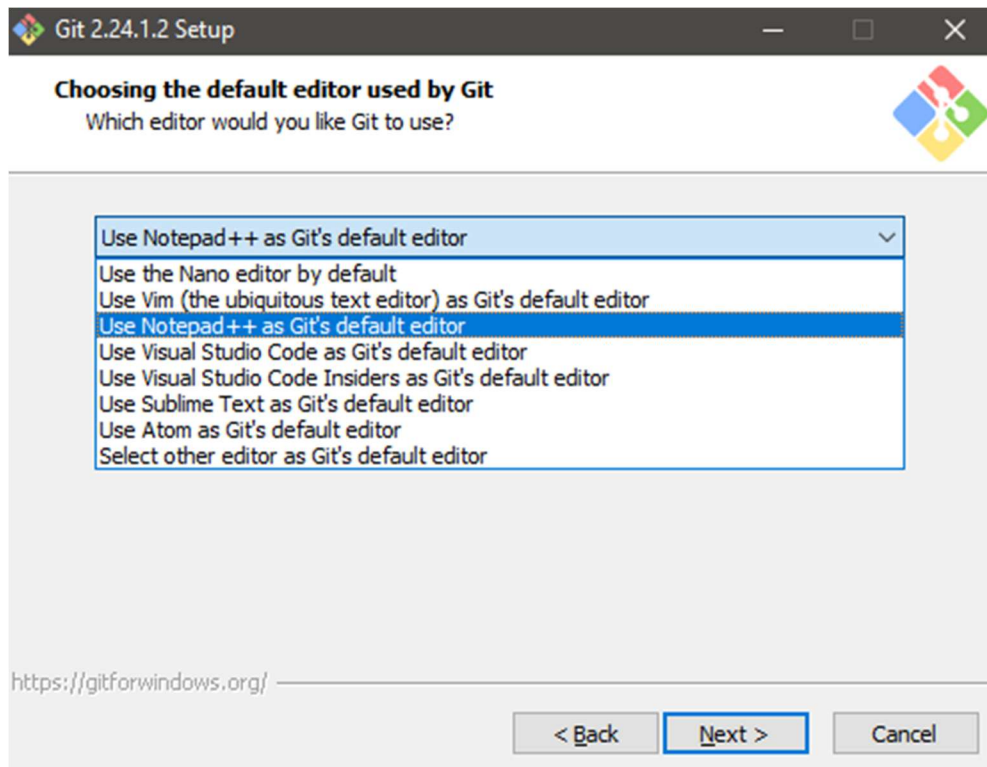
1. A feature in our group project suddenly broke, but we were able to look at our code history and fix what broke it.
2. It makes it easier to keep track of the contribution of my group mates. This is especially helpful when you need a way to quantify contribution per member.
3. Do you ever feel afraid to change code in your project? I mess up a lot, but I can easily reset any changes I make. I also work on a separate “copy” of a project when I'm adding a new feature.

Installing Git

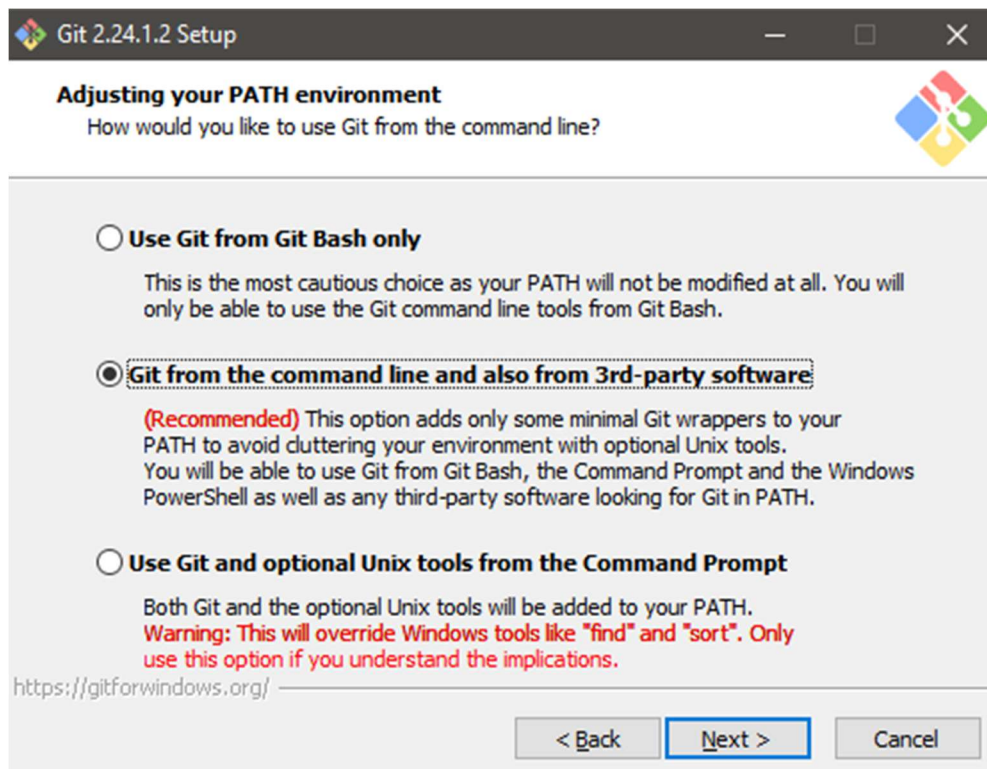
To install Git on Windows, go to <https://git-scm.com/download/win>.

You can keep most of the default settings, but I recommend changing the default editor to something you're more comfortable with:





And also allow using Git from the command line, if you prefer using this over Git Bash.



Git Bash is kind of like a command line, but for Git. Here's a side by side comparison of the two.

<pre>limri@Riana-Lim MINGW64 ~ \$ cd "C:\Users\limri\Documents\GitHub Tutorials" limri@Riana-Lim MINGW64 ~/Documents/GitHub Tutorials (master) \$ git status On branch master Your branch is up to date with 'origin/master'. nothing to commit, working tree clean</pre>	<pre>Microsoft Windows [Version 10.0.18362.535] (c) 2019 Microsoft Corporation. All rights reserved. C:\Users\limri>cd C:\Users\limri\Documents\GitHub Tutorials C:\Users\limri\Documents\GitHub Tutorials>git status On branch master Your branch is up to date with 'origin/master'. nothing to commit, working tree clean</pre>
---	---

Git Bash on the left, Windows CMD on the right

After installing, open your command line (or bash) and run `git config --global user.name <your name>` to set up your Git username. Adding `--global` will apply this email address of the user of your system. If you just want to set-up the email for the specific repository you're working on, you can change this to `--local`.

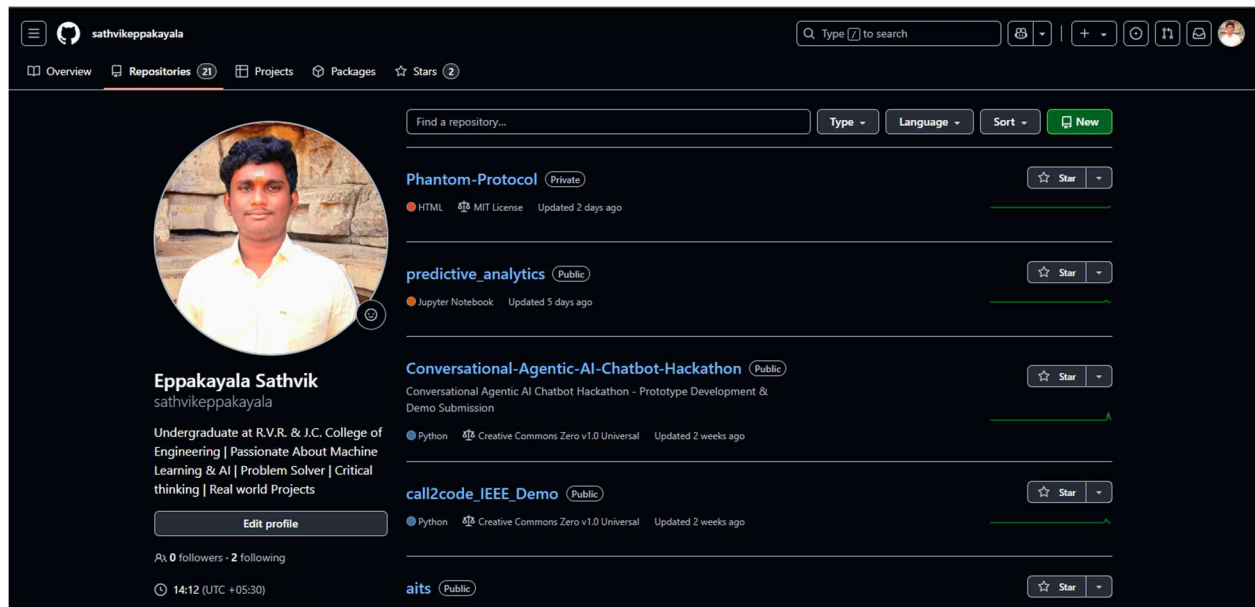
After setting up your Git username, run `git config --global user.email "your_email@example.com"` to set-up Git to use your email.

Creating a Repository on GitHub

I will assume that you've created an account on GitHub (it isn't any different from creating an account on other websites). **If you are from the Ateneo, you can add your OBF email to your account for a pro account.**

In GitHub, projects are called *repositories*. You can think of repositories as “folders” where all your project files are. The repository is where you do all the version control stuff.

First, you need to create a repository. You can find the “New ” button above your list of repositories in the home page, or when you navigate to your profile and click on “Repositories”.



Look out for a green button that says “New”

You’ll be taken to this page:

There are two possible ways to move forward depending on what you want to do:

1. Pushing into GitHub repository

Since you're moving a project you already have into a GitHub repository, don't tick the box that says "Initialize this repository with a **README**".

You'll be taken to a page that looks like this:

To turn a project you already have into a repository, open the terminal and move to your project's folder. To do this, run the command `cd <directory address>`. (CD means Change Directory)

```
rvr@DESKTOP-DI48LBN MINGW64 ~ (master)
$ git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
          [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--no-lazy-fetch]
          [--no-optional-locks] [--no-advice] [--bare] [--git-dir=<path>]
          [--work-tree=<path>] [--namespace=<name>] [--config-env=<name>=<envvar>]
          <command> [<args>]

These are common Git commands used in various situations:


start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one


work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index


examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  diff       Show changes between commits, commit and working tree, etc
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status
```

Run the command `git init`. This will *initialize* a Git repository.

```
C:\Users\limri\Documents\GitHub Tutorials>git init
Initialized empty Git repository in C:/Users/limri/Documents/GitHub Tutorials/.git/
```

Run the command `git add .` The “.” means that you're adding all the created, changed, or deleted files that are in your folder to the commit you're making. You can think of a commit as like save states in video games. You changed something in your repository, so you wanna save the state it's in so you could go back to it later on if you need to. **Note that it is bad practice to add all files at once, but since we are learning how to create repositories first we can let it slide for now.**

Next, run the command `git commit -m "<your commit message>".` Let's dissect that for a bit. "Commit" tells Git that you're making a commit. "-m" means message which tells git that the following line in quotation marks is your commit message. I'll explain commits later on, but for now, just explain what you changed or uploaded.

```
C:\Users\limri\Documents\GitHub Tutorials>git add .

C:\Users\limri\Documents\GitHub Tutorials>git commit -m "Add files ABC057D.java, ABC05D.java, and ABC114D.java"
[master (root-commit) fee919a] Add files ABC057D.java, ABC05D.java, and ABC114D.java
3 files changed, 160 insertions(+)
 create mode 100644 ABC057D.java
 create mode 100644 ABC05D.java
 create mode 100644 ABC114D.java
```

Here's what it looks like in my terminal!

Copy the commands under the section that says "...or push an existing repository from the command line" (this is what we're doing!) and run them in your terminal.

```
git remote add origin <your repository's link>
git push origin -u master
```

```
C:\Users\limri\Documents\GitHub Tutorials>git remote add origin https://github.com/rairayy/github-tutorial-2.git

C:\Users\limri\Documents\GitHub Tutorials>git push -u origin master
fatal: HttpRequestException encountered.
An error occurred while sending the request.
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.97 KiB | 1.97 MiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://github.com/rairayy/github-tutorial-2.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Here's what it looks like in my terminal!

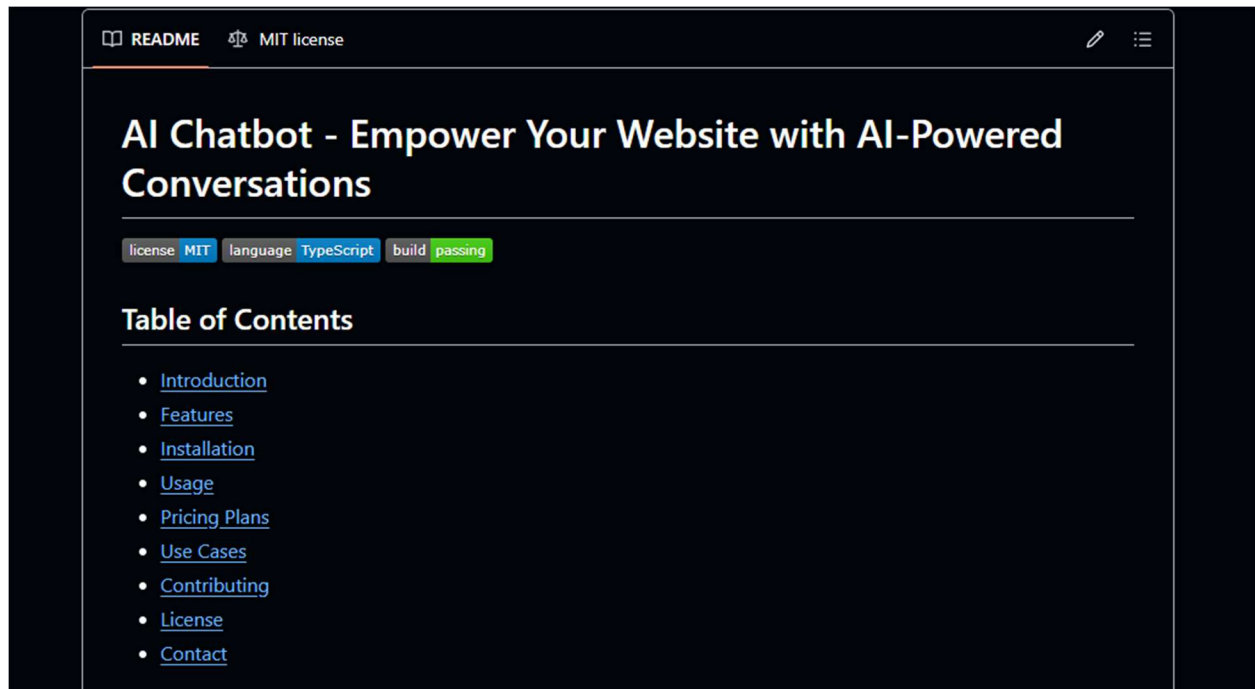
If you go back to your repository on GitHub, your latest commit be there!

The screenshot shows the GitHub interface for a repository named 'aoldev' (Public). At the top, there are buttons for 'Edit Pins', 'Watch' (0), 'Fork' (3), and 'Star' (3). Below this, the repository is shown with 'main' as the selected branch, 7 branches, and 0 tags. A search bar and 'Add file' button are present. The main content area displays a list of files and folders with their commit history:

File/Folder	Description	Commit Date
aoldevceo	Merge pull request #64 from aoldev/dependabot/npm_and_yarn/multi-08c...	51baceb · 5 months ago
.github/ISSUE_TEMPLATE	Update issue templates	6 months ago
Connection	Done	6 months ago
app	Add view job position page and apply for job page	6 months ago
components/ui	Add files via upload	6 months ago
hooks	UI	6 months ago
lib	UI	6 months ago
models	Done	6 months ago
public	Add files via upload	6 months ago

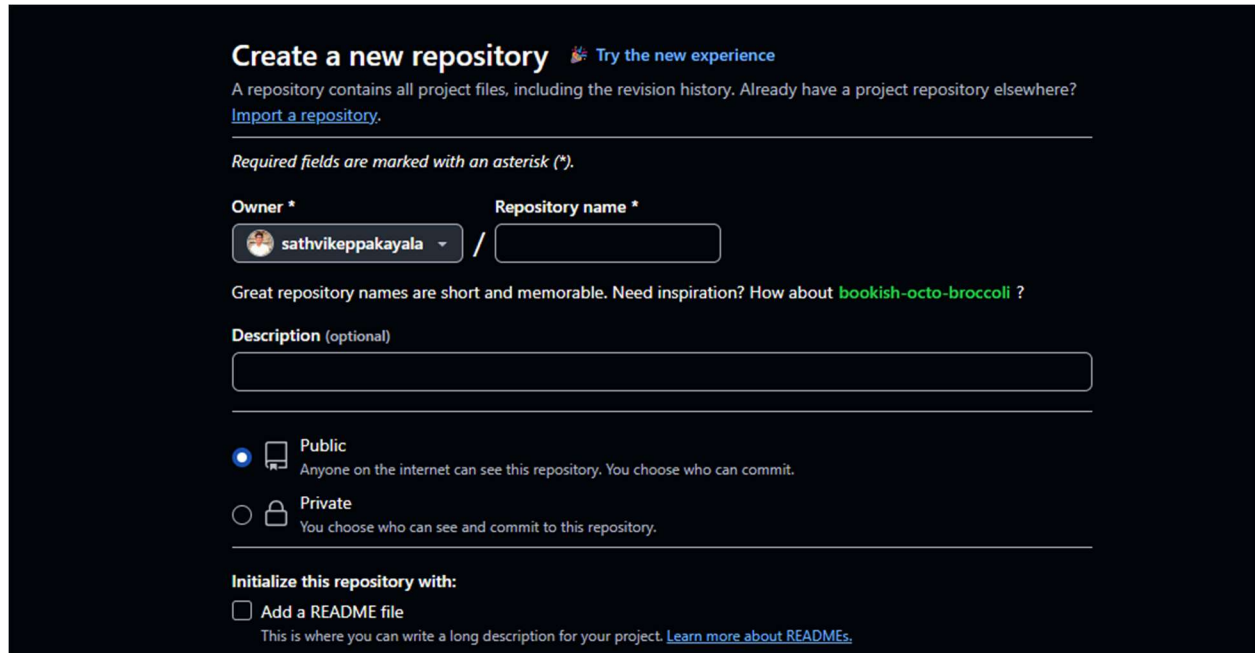
On the right side, there is an 'About' section with a description: 'No description, website, or topics provided.' Below this are links for 'Readme', 'MIT license', 'Activity', 'Custom properties', '3 stars', '0 watching', '3 forks', and 'Report repository'. At the bottom, there is a 'Releases' section with the text 'No releases published' and a link to 'Create a new release'.

GitHub will also suggest that you add a README to your project. It'll look something like this when you've made one. When people visit your repository, the README is what they'll see right after your files so it's best to explain things there such as a more detailed description of your project or what dependencies your project has. (Dependencies are things people will need to install to make your project run on their devices, aka what other technologies your project *depend* on)



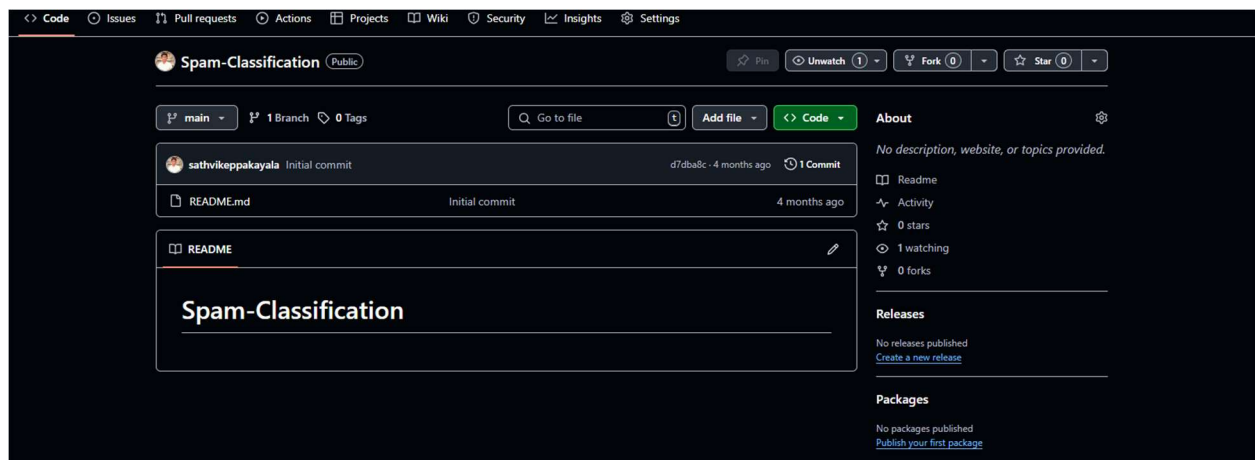
2. Starting a project in GitHub

Since you're starting a project from scratch, you can tick the box that says "Initialize this repository with a README" and create a repository.



The screenshot shows the 'Create a new repository' form on GitHub. At the top, it says 'Create a new repository' with a link to 'Try the new experience'. Below this is a description of a repository and a link to 'Import a repository'. A note states 'Required fields are marked with an asterisk (*)'. The form has two main sections: 'Owner' and 'Repository name', both marked with an asterisk. The 'Owner' field has a dropdown menu showing 'sathvikeppakayala'. The 'Repository name' field is empty. Below these fields is a suggestion: 'Great repository names are short and memorable. Need inspiration? How about bookish-octo-broccoli?'. The 'Description' field is optional and empty. There are two radio buttons for visibility: 'Public' (selected) and 'Private'. Below these is the 'Initialize this repository with:' section, which has a checkbox for 'Add a README file' (selected). A link 'Learn more about READMEs.' is provided.

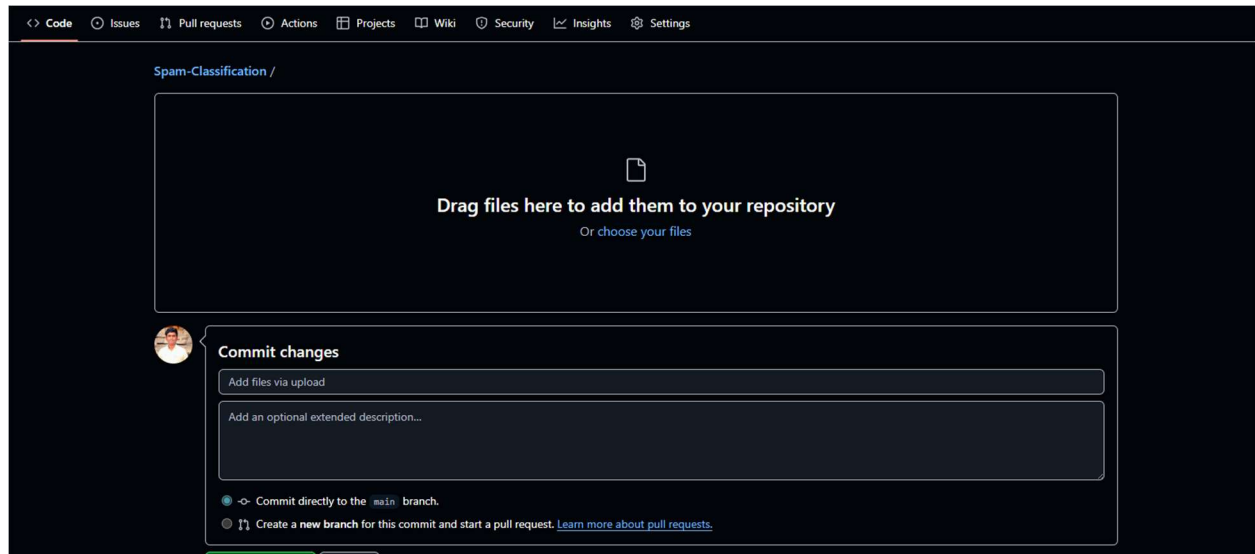
You will be taken to a page that looks like this:



If you want to upload or create files using the website, you can simply press the "Create new file" or "Upload files" buttons. Either way, you'll be asked to "Commit changes" after.

It will also let you choose between committing to the master branch or creating a new branch. You can think of branching like creating a copy of the current state of your repository, but I will explain this later on as well. If you're just using a practice repository, you can commit to the

master branch for now. **Note that we don't usually commit to the master branch, we usually make another branch first.**



If you want to work on your terminal, move to the folder you'd like to work in and run the command `git init` to initialize a repository.

After you've made some changes, you can run these commands which I've discussed in #1.

```
git add .
git commit -m <your commit message>
git remote add origin <your repository's link>
git push origin -u master
```

Clone a GitHub Repository

1. Initially you need to click the options repository on GitHub.
2. Then in the top right corner, click the option clone or download where a small drop-down box will appear having a URL for cloning over HTTPS.
3. Then enter into your Powershell windows and write clone URL as:
`git clone repository_url`
4. On the other hand, you can clone a github repository with SSH URLs where first you need to generate an SSH key pair on your windows workstation as well as need to assign a public key to your GitHub account.

List Remote Repositories

1. Make a copy of the repository from GitHub for your working directory.

2. Ensure that the working directory should have the project name as "cd git_project" and replace the project name from the downloaded repository.
3. If the above option doesn't work, you can list the content using "ls command" for the current directory, especially to check your exact number of spellings.
4. Besides, you can list the remote repository in the sub-directory as "git remote -v".

Using Git Commands:

- **git init:** Initializes a new Git repository.
- **git clone <repository>:** Clones an existing repository.
- **git status:** Displays the status of your working directory and staging area.
- **git add <file>:** Adds a file to the staging area.
- **git commit -m "message":** Commits the changes in the staging area with a message.
- **git push:** Pushes changes to a remote repository.
- **git pull:** Pulls changes from a remote repository.
- **git log:** "[git log](#)" command will help you to see the logs of all the commits made.
- **git branch:** Creates a new branch.
- **git merge:** For merging the changes from one branch to another branch.
- **git config:** "[git config](#)" will help you to configure the username and email id.
- **git tag:** It will display the all tags.

create a new repository

create a new directory, open it and perform a

git init

to create a new git repository.

checkout a repository

create a working copy of a local repository by running the command

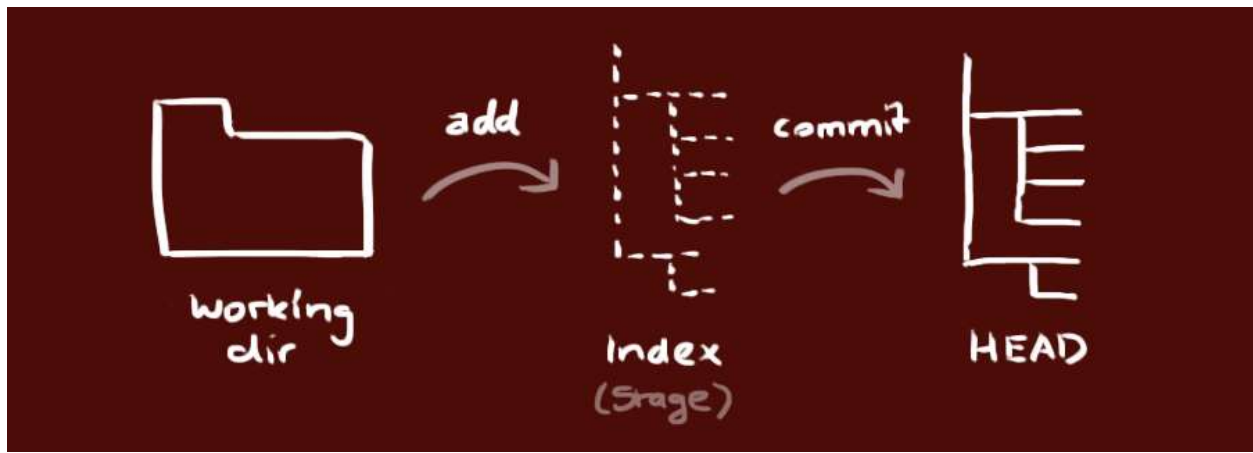
`git clone /path/to/repository`

when using a remote server, your command will be

`git clone username@host:/path/to/repository`

workflow

your local repository consists of three "trees" maintained by git. the first one is your Working Directory which holds the actual files. the second one is the Index which acts as a staging area and finally the HEAD which points to the last commit you've made.



add & commit

You can propose changes (add it to the **Index**) using

`git add <filename>`

`git add *`

This is the first step in the basic git workflow. To actually commit these changes use

`git commit -m "Commit message"`

Now the file is committed to the **HEAD**, but not in your remote repository yet.

pushing changes

Your changes are now in the **HEAD** of your local working copy. To send those changes to your remote repository, execute

`git push origin master`

Change *master* to whatever branch you want to push your changes to.

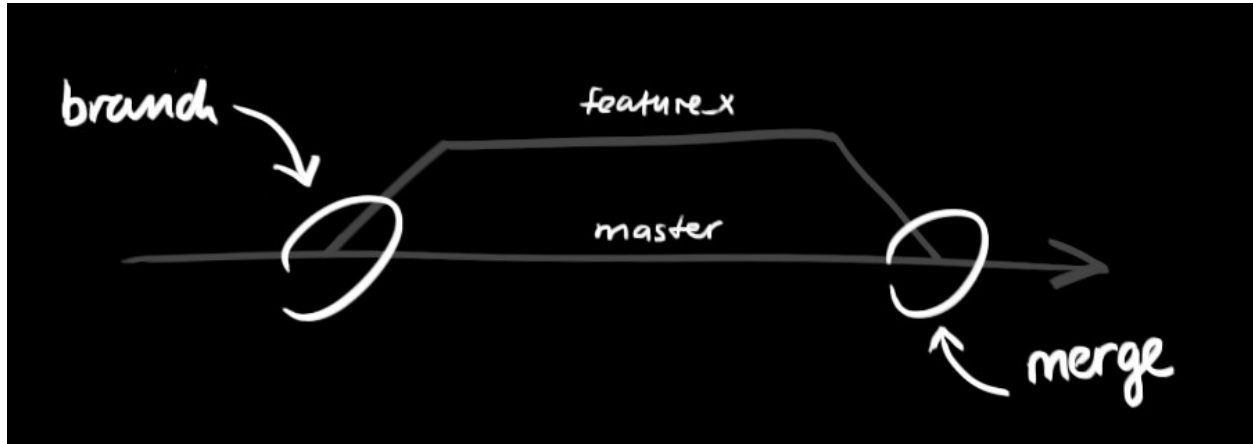
If you have not cloned an existing repository and want to connect your repository to a remote server, you need to add it with

`git remote add origin <server>`

Now you are able to push your changes to the selected remote server

branching

Branches are used to develop features isolated from each other. The *master* branch is the "default" branch when you create a repository. Use other branches for development and merge them back to the master branch upon completion.



create a new branch named "feature_x" and switch to it using

`git checkout -b feature_x`

switch back to master

`git checkout master`

and delete the branch again

`git branch -d feature_x`

a branch is *not available to others* unless you push the branch to your remote repository

`git push origin <branch>`

update & merge

to update your local repository to the newest commit, execute

`git pull`

in your working directory to *fetch* and *merge* remote changes.

to merge another branch into your active branch (e.g. master), use

`git merge <branch>`

in both cases git tries to auto-merge changes. Unfortunately, this is not always possible and results in *conflicts*. You are responsible to merge those *conflicts* manually by editing the files shown by git. After changing, you need to mark them as merged with

`git add <filename>`

before merging changes, you can also preview them by using

`git diff <source_branch> <target_branch>`

tagging

it's recommended to create tags for software releases. this is a known concept, which also exists in SVN. You can create a new tag named *1.0.0* by executing

```
git tag 1.0.0 1b2e1d63ff
```

the *1b2e1d63ff* stands for the first 10 characters of the commit id you want to reference with your tag. You can get the commit id by looking at the...

log

in its simplest form, you can study repository history using.. `git log`

You can add a lot of parameters to make the log look like what you want. To see only the commits of a certain author:

```
git log --author=bob
```

To see a very compressed log where each commit is one line:

```
git log --pretty=oneline
```

Or maybe you want to see an ASCII art tree of all the branches, decorated with the names of tags and branches:

```
git log --graph --oneline --decorate --all
```

See only which files have changed:

```
git log --name-status
```

These are just a few of the possible parameters you can use. For more, see `git log --help`

replace local changes

In case you did something wrong, which for sure never happens ;), you can replace local changes using the command

```
git checkout -- <filename>
```

this replaces the changes in your working tree with the last content in HEAD. Changes already added to the index, as well as new files, will be kept.

If you instead want to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it like this

```
git fetch origin
```

```
git reset --hard origin/master
```